Отображение графовой модели данных в каноническую объектно-фреймовую информационную модель при создании систем интеграции неоднородных информационных ресурсов*

© С. А. Ступников
ИПИ РАН,
Москва
ssa@ipi.ac.ru

Аннотация

В работе рассматривается отображение модели данных атрибутированных графов в каноническую информационную модель данных для создания систем виртуальной или материализованной интеграции неоднородных информационных ресурсов – СУБД, Веб-сервисов и т.д.

Целью работы является создание обоснованной теоретической базы для интеграции ресурсов, основанных на графовых моделях.

Работа выполнена при поддержке РФФИ (гранты 11-07-00402-а, 13-07-00579-а) и Президиума РАН (программа 16П, проект 4.2).

1 Введение

Роль данных в различных областях деятельности человека – научных исследованиях, здравоохранении, образовании, промышленности и т.д. – непрерывно растет в последние годы. Укрепляется новая парадигма в науке и информационных технологиях, связанная с интенсивным использованием данных – так называемая *четвертая парадигма* [16]. Развиваются новые информационные технологии, в которых данные становятся доминирующим фактором, новые подходы к концептуализации, организации и реализации информационных систем. При этом требуется не только создание методов и средств оперирования данными, объемы которых выходят за рамки возможностей современных технологий баз данных, но и разработка новых подходов, позволяющих справляться с разнообразием массово и хаотично развивающихся языков и моделей данных.

Данная статья продолжает исследования по унификации моделей, применяемых в системах с интенсивным использованием данных, для виртуальной или материализованной интеграции ресурсов при создании федеративных баз данных или хранилищ данных. К таким моделям относятся разнообразные NoSQL-модели; онтологические и семантические модели; графовые модели; модели, основанные на многомерных массивах и т.д.

Материализованная интеграция предполагает создание хранилища данных, в которое загружаются интегрируемые ресурсы. При этом данные из схемы ресурса преобразуются в общую схему хранилища.

Виртуальная интеграция обычно предполагает создание *предметных посредников,* образующих промежуточный слой между пользователем (приложением) и неоднородными информационными ресурсами. Данные из ресурсов не материализуются в посредниках: доступ к данным осуществляется при помощи запросов к посреднику в терминах федеративной схемы посредника. Эти запросы переписываются в частичные запросы над информационными ресурсами, затем исполняются на ресурсах. Результаты частичных запросов объединяются и выдаются пользователю также в терминах федеративной схемы [8].

Унификацией модели данных ресурса называется ее отображение в *каноническую информационную модель* (служащую общим языком в среде разнообразных моделей ресурсов), сохраняющее информацию и семантику операций языка манипулирования данными (ЯМД) [20]. Унификация моделей ресурсов является необходимым предусловием материализованной или виртуальной интеграции ресурсов, т.к. семантические отображения, связывающие федеративную схему и схемы ресурсов, нужно проводить в единой (канонической) модели [9].

Труды 15-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2013, Ярославль, Россия, 14-17 октября 2013 г.

В ранее проведенных исследованиях изучались вопросы унификации NoSQL-моделей [21] и моделей, основанных на многомерных массивах [22].

В данной статье рассматривается еще один важный класс моделей данных – *графовые модели*. Исследования графовых моделей начались в середине 1980-х годов. Математическими основаниями для них послужила теория графов, а наибольшее влияние оказали так называемые *семантические модели* (например, модель «сущность-связь») [3]. Целью графовых моделей было преодоление ограничений, налагаемых традиционными моделями данных, связанных с представлением исходных графовых структур данных.

Основными отличительными чертами графовых моделей данных являются следующие [3]:

- данные и/или схема данных представляются в виде графов или структур данных, обобщающих понятие графа (гиперграфы или гипервершины);
- манипулирование данными выражается в виде трансформаций графов или при помощи операций, основными параметрами которых являются такие характерные графовые структуры и свойства, как пути, подграфы, связность и т.д.;
- ограничения целостности тесно связаны с графами как структурой данных. Так, ограничениями могут быть уникальность меток ребер и вершин, типизация ребер и вершин, ограничения на область определения и область значений свойств ребер и вершин.

Графовые модели данных применяются в тех случаях, когда информация о взаимосвязях между данными или их топологии является более важной (или настолько же важной), как сами данные. Поводом к использованию графовых моделей может быть также недостаточная выразительная сила языков запросов традиционных моделей. Наиболее распространенными примерами применения графовых моделей являются системы управления и анализа сложных сетей – социальных, биологических, информационных, транспортных, телекоммуникационных и других.

Наибольшего разнообразия в своем развитии графовые модели достигли в 1990-х годах. Наряду с обычными графами, представляющими собой множества вершин (помеченных или непомеченных), соединенных ребрами (направленными или ненаправленными, помеченными или непомеченными), развитие в графовых моделях получили такие структуры, как *гипервершины* и *гиперграфы*. Гипервершина представляет собой вложенный граф, а ребра гиперграфа могут соединять не две, а произвольное количество вершин [7].

Однако, обзоры состояния современных графовых СУБД [3] показывают, что большинство существующих баз данных основаны на простых или атрибутированных графах (*attributed graph* или *property graph*), в которых атрибуты (свойства) приписываются ребрам и/или вершинам графа. Именно такие модели и были выбраны в данной статье в качестве исходных, подлежащих унификации моделей.

В качестве канонической модели в работе рассматривается объектно-фреймовая модель данных, а именно - язык СИНТЕЗ [10], нацеленный на разработку предметных посредников для решения задач в средах неоднородных ресурсов, и поддержанный программными средствами исполнительной среды предметных посредников.

Статья организована следующим образом.

В разделе 2 рассмотрена и проиллюстрирована на примере модель данных атрибутированных графов.

В разделе 3 рассмотрены и проиллюстрированы основные принципы отображения модели данных атрибутированных графов в язык СИНТЕЗ.

В разделе 4 рассмотрены вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN [1].

В разделе 5 рассмотрены родственные исследования и направления дальнейшей работы.

2 Модель данных атрибутированных графов

В настоящее время существует большое количество СУБД, модели данных которых основаны на простых или атрибутированных графах. Языки определения данных (ЯОД), языки манипулирования данными (ЯМД), прикладные интерфейсы пользователя (API) различаются в этих системах весьма существенно. Для того, чтобы обеспечить общность подхода по унификации графовых моделей, в данной статье рассматривается синтетическая модель данных атрибутированных графов. Структуры данных модели покрывают возможности моделей таких известных систем, как, например, Neo4j [11], Dex [15], InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton, InfoGrid.

В качестве ЯМД синтетической модели рассматривается декларативный язык Cypher [17], развиваемый в системе Neo4j. Поэтому, фактически, рассматриваемая модель является расширением графовой модели Neo4j. С точки зрения общности по отношению к другим графовым языкам запросов, язык Cypher покрывает такие классы возможностей, как смежность (adjacency) вершин и ребер, достижимость по путям фиксированной длины (fixed-length paths reachability), достижимость по простым регулярным путям (regular simple paths), поиск кратчайших путей, поиск подграфов по образцу (pattern matching) [3]. Это также означает,

что язык Cypher покрывает возможности языков запросов основных современных графовых баз данных (в том числе, перечисленных в предыдущем параграфе).

Итак, синтетическая модель выбрана таким образом, чтобы рассматриваемые методы отображения ее в каноническую (раздел 3) можно было применить для унификации различных реальных графовых моделей систем, упомянутых выше.

Заметим, что в данной работе не рассматривается важный класс СУБД, основанных на модели RDF [12], включающий такие системы, как AllegroGraph, G-Store, BrightstarDB. Часто RDF относят к графовым моделям. Однако, ввиду обширности области применения и развития приложений RDF, а также специфики ЯОД, ЯМД и семантики RDF по сравнению с основной массой графовых моделей, вопросы унификации RDF следует рассматривать отдельно.

Рассмотрим сначала вопросы определения данных в модели данных атрибутированных графов.

База данных в модели есть граф, вершины и ребра которого *типизированы*. Тип вершины или ребра представляет собой, фактически, совокупность атрибутов (свойств), приписываемых вершине или ребру. Определим формально множество всевозможных типов вершин *VertexTypes* и множество типов ребер *EdgeTypes*.

Так, *VertexTypes* представляет собой множество троек вида <*id, name, A*>, где *id* – идентификатор типа (например, целое число), *name* – имя типа (строка символов), *A* – подмножество множества всевозможных атрибутов *Attributes*.

Множество атрибутов *Attributes* есть множество кортежей вида <*id, name, type*>, где *id* и *name* – идентификатор и имя атрибута соответственно, *type* $\in$ *B* – тип атрибута, *B* – множество встроенных типов (например, *boolean, int, float, string,*типы массивов и т.д.)

Множество *EdgeTypes* есть набор кортежей вида <*id, name, A, directed, restricted, head, tail*>, где *id* – идентификатор типа; *name* – имя типа; *A* $\subseteq$ *Attributes*; *directed* $\in$ *{true, false}* – флаг направленности ребра; *restricted* $\in$ *{true, false}* – булевский флаг определенности типов вершин, которые связывает ребро; *head* $\in$ *VertexTypes* – тип исходящей вершины ребра; *tail* $\in$ *VertexTypes* – тип входящей вершины ребра.

Для любого типа *T* $\in$ *EdgeTypes* если *restricted(T) = true*, то значения *head(T), tail(T)* определены; если же *restricted(T) = false*, то значения *head(T), tail(T)* не определены.

Произвольная *схема S = <VT(S), ET(S)>* обобщенной графовой модели включает два множества: множество типов вершин *VT(S)* $\subseteq$ *VertexTypes* и множество типов ребер *ET(S)* $\subseteq$ *EdgeTypes.*

База данных (граф) *G*, удовлетворяющий схеме *S* имеет вид *G = <V, E>*, где

- *V = {v | ∃T.(T ∈ VT(S) & v: T)}* – множество вершин графа такое, что любая вершина имеет тип из *VT(S)*;
- *E = {<e, t, h> | ∃T.(T ∈ ET(S) & e: T & t ∈ V & t: tail(T) & h ∈ V & h: head(T))}* – множество ребер графа такое, что любое ребро имеет тип из *ET(S)* и соединяет вершины из *V*.

Типизация *x: T* означает, что для вершины (ребра) *x* могут быть определены атрибуты из *A(T)* (атрибуты типа *T*).

Рассмотрим пример схемы *Cinema* базы данных фильмов [5] в обобщенной модели. Для упрощения будем опускать в примерах идентификаторы типов и атрибутов, считая, что имена однозначно идентифицируют типы:

VT(Cinema) = { people, movie }
ET(Cinema) = { cast, directs }
A(movie) = {<id, long>, <title, string>, <year, integer>}
A(people) = {<id, long>, <name, string>}
cast = <{<character, string>}, false, false, undefined, undefined>
directs = <∅, true, true, people, movie>

Схема включает два типа вершин (*people, movie*) и два типа ребер (*cast, directs*). Тип *movie* включает три атрибута (*id, title, year*), *people* – два (*id, name*), *cast* – один (*character*), *directs* – ни одного. Ребра типа *cast* являются ненаправленными и типы вершин, которые они связывают, не определены; ребра типа *directs* – направлены от вершины типа *people* к вершине типа *movie*.

Пример простого графа *g*, удовлетворяющего схеме *Cinema* выглядит следующим образом:

g = <{m, p}, {e}>
m: movie = <id: 1, title: “Lost in Translation”, year: 2003>
p: people = <id: 1, name: “Scarlett Johansson”>
e: cast = <<character: “Charlotte”>, m, p>

Вопрос выбора ЯМД для обобщенной графовой модели достаточно сложен. Графовые языки запросов развивались в течении многих лет вместе с самими графовыми моделями [2]. Существуют работы по анализу и сравнению выразительной силы и вычислительной сложности графовых ЯМД [19].

Однако, в современных популярных графовых СУБД языки манипулирования представлены в большинстве случаев просто прикладным интерфейсом пользователя (API), предоставляющим доступ к структуре графа, методы обхода графа и инкапсулирующим основные алгоритмы на графах. Не существует стандарта графового языка запросов.

В данной работе в качестве ЯМД модели атрибутированных графов выбран язык Cypher [17]. С одной стороны, этот язык поддерживается и развивается в одной из самых популярных графовых СУБД с открытым кодом – Neo4j. С другой стороны, язык является декларативным, в отличие от существующих графовых API или скриптовых языков (как Gremlin). Язык основывается на различных подходах и сложившихся техниках выразительных запросов. Основные конструкции и ключевые слова имеют сходство с такими широко распространенными языками, как SQL и SPARQL.

3 Отображение модели атрибутированных графов в каноническую информационную модель

В качестве канонической модели в данной статье рассматривается объектная модель языка СИНТЕЗ [10]. Объектные модели хорошо зарекомендовали себя при унификации различных классов моделей – структурированных, онтологических, сервисных, процессных [8]. Поэтому есть основания выбирать канонические объектные модели при интеграции информационных ресурсов, представленных в моделях различных классов. При этом графовые модели выступают как один из классов моделей ресурсов, подлежащих интеграции.

3.1 Отображение языка определения данных

Схема в обобщенной графовой модели представляется в языке СИНТЕЗ в виде одноименного модуля (например, *Cinema*), включающего классы, содержащие вершины и ребра графа (например, *vertices* и *edges* соответственно):

```
{ Cinema; in: module;
  { vertices; in: class; … },
  { edges: in: class; … };
  …
}
```

Тип вершины (например, *movie* – см. раздел 2) представляется в языке СИНТЕЗ одноименным классом (который объявляется подклассом класса всех вершин *vertices*), также входящим в модуль, соответствующий схеме:

```
{ Movie; in: class; superclass: vertices;
  instance_type: {
    id: long;
    title: string;
    year: integer; };
}
```

Атрибуты типа вершины, представляются в языке СИНТЕЗ атрибутами типа экземпляров (*instance_type*) соответствующего класса. Между встроенными типами графовой модели (long, string, int и т.д.) и встроенными типами языка СИНТЕЗ (long, string, integer) устанавливается взаимно-однозначное соответствие.

Тип ребра (например, *directs* – см. раздел 2) также представляется одноименным классом (который объявляется подклассом класса всех вершин *edges*):

```
{ directs; in: class; superclass: edges;
  instance_type: {
    metaframe
      directed: true;
      restricted: true;
      startVertexType: people;
      endVertexType: movie;
    end
    edgeConstr: {in: invariant;
      {{ all e/directs.inst (directs(e) ->
           people(e.startVertex) & movie(e.endVertex)) }}
    }; };
}
```

Атрибуты типа ребра, аналогично типу вершины, представляются в языке СИНТЕЗ атрибутами типа экземпляров соответствующего класса.

Заметим, что информация о направленности (*directed*), определенности ребра (*restricted*), типах его исходящей (*startVertexType*) и входящей (*endVertexType*) вершин представляется специальной конструкцией - метафреймом [9], связанным с типом экземпляра класса. Метафреймы в языке СИНТЕЗ предназначены для выражения дополнительной метаинформации, связанной с такими сущностями, как модули, типы, классы, функции.

Кроме того, ограничение на типы исходящей и входящей вершин представляется инвариантом *edgeConstr*, заданным формулой в типизированной логике первого порядка. Знак *all* означает квантор всеобщности, знак -> - логическую импликацию, & - конъюнкцию, выражение *x/T* – типизацию переменной *x* типом *T*, *C.inst* – тип экземпляров (instance) класса *C*. Предикат *C(x)*, где *C* – имя класса, обращается в истину на экземплярах класса *C*.

Заметим также, что на переменной *e* типа *directs.inst* определены атрибуты исходящей вершины ребра *startVertex* и входящей вершины ребра *endVertex*, хотя их нет непосредственно в типе *directs.inst*. Эти атрибуты являются общими для всех типов вершин и принадлежат типу экземпляров класса *edges*:

```
{ edges; in: class;
  instance_section: {
    startVertex: vertices.inst;
    endVertex: vertices.inst;
    isValidEdge: { in: predicate;
      params: {+stVtx/vertices.inst,
               +endVtx/vertices.inst
               returns/Boolean };
      {{ (stVtx = this.startVertex &
          endVtx = this.endVertex -> returns = true) &
         (stVtx <> this.startVertex |
          endVtx <> this.endVertex -> returns = false) }}
    };
}
```

Кроме упомянутых атрибутов, тип *edges.inst* включает метод-предикат *isValidEdge*. Предикат

e.isValidEdge(v1, v2) обращается в истину, если исходящая вершина ребра *e* (*e.startVertex*) совпадает с *v1* и входящая вершина ребра *e* (*e.endVertex*) совпадает с *v2*. Спецификация метода задается формулой первого порядка, связывающей входные и выходные параметры метода. Знак | означает дизъюнкцию, <> - неравенство, *this* – объект, для которого вызывается метод.

3.2 Отображение языка манипулирования данными

При интеграции неоднородных ресурсов (баз данных, сервисов и т.д.) необходимо отображение ЯОД модели ресурса в каноническую. ЯМД канонической модели, напротив, необходимо отображать в ЯМД модели ресурса, т.к. запросы к посреднику в канонической модели нужно отображать в запросы к ресурсам.

Язык запросов (программ) модели СИНТЕЗ представляет собой Datalog-подобный язык в объектной среде. Программа представляет собой набор конъюнктивных запросов (правил) вида

$q(x/T) :- C_1(x_1/T_1), \dots, C_n(x_n/T_n),$
$F_1(X_1, Y_1), \dots F_m(X_m, Y_m), B.$

Тело запроса представляет собой конъюнкцию предикатов-коллекций, функциональных предикатов и ограничения. Здесь C_i - имена коллекций (классов), F_j – имена функций, x_i – имена переменных, значения которых пробегают по классам, T_i – типы переменных, X_j и Y_j – входные и выходные параметры функций, B – ограничение, налагаемое на x_i, X_j, Y_j.

В дальнейшем будет использоваться запись предиката-коллекции вида *movie([title, year])*. Неформально это означает, что нас не интересуют объекты класса *movie* целиком, а лишь их атрибуты *title*, *year*. Формально запись означает сокращение от *movie(_/movie.inst[title, year])*. Здесь знак _ обозначает анонимную переменную, *movie.inst* – анонимный тип экземпляров (instance) класса *movie*, *title, year* – необходимые атрибуты типа экземпляров класса.

Будет также использоваться запись *source([i, j, val1/val])*, означающая переименование атрибута *val* в *val1*.

Ввиду ограниченного объема статьи, отображение основных конструкций ЯМД будет продемонстрировано на нескольких примерах.

Пример 1 (Конъюнктивный запрос с использованием предиката смежности вершин и ребер). Рассмотрим запрос, возвращающий имена актеров по фамилии Круз, игравших в фильмах вместе со Скарлетт Йохансон:

```
q([colleague_name]) :-
people(scarlett/[name]),
movies(m),
people(colleague/[collegue_name: name]),
cast(c1), cast(c2),
c1.isValidEdge(m, scarlett),
c2.isValidEdge(m, colleague),
scarlett.name = “Scarlett Johansson”,
colleague.name.like(“*Cruz*”).
```

Запрос вернет непустой результат, если в графе базы данных существуют такие вершины-фильмы *m*, и такие ребра *c1, c2* типа *cast*, что *c1* соединяет *m* с вершиной *scarlett*, и *c2* соединяет *m* с вершиной *colleague*.

В языке Cypher такой запрос имеет вид

```
START scarlett =
  node:node_auto_index(name = 'Scarlett Johansson')
MATCH m-[c1:cast]-scarlett, m-[c2:cast]-collegue
WHERE collegue.name =~ /*Cruz*/
RETURN collegue.name
```

Каждый запрос языка Cypher представляет собой образец (pattern), по которому производится поиск в графе базы данных.

В секции START запроса указываются вершины или ребра, с которых следует начинать поиск. В данном случае это вершина *scarlett*, поскольку для нее указано значение атрибута *name*, а значит, возможен поиск по индексу этого атрибута.

В секции MATCH указывается образец поиска в графе, привязанный к стартовым вершинам. В данном случае это указание, что следует искать фильмы, в которых играла (*Cast*) *scarlett*, а также других актеров, играющих в том же фильме.

В секции WHERE указывается фильтр поиска. В данном случае это фамилия коллеги-актрисы.

В секции RESULT указываются возвращаемые значения. В данном случае это полное имя коллеги-актера.

Основные принципы отображения конъюнктивных запросов объектной модели в язык Cypher, проиллюстрированные на данном примере, состоят в следующем:

- конъюнктивный запрос представляется в языке Cypher запросом, возвращающим результат (секция RETURN);
- предикаты-коллекции и предикат смежности вершин и ребер представляются образцами секции MATCH. Каждому предикату смежности соответствует свой образец. Переменные, типизированные в предикатах-коллекциях, представляются одноименными переменными, использующимися в образцах;
- предикаты-условия представляются соответствующими предикатами секции WHERE или START;
- атрибуты результирующего предиката конъюнктивного запроса представляются одноименными атрибутами в секции RETURN.

Пример 2 (Удаление вершин). Рассмотрим запрос, удаляющий из базы данных фильм «Отчаянный»:

-movie(m) :- movie(m), m.year = “Desperado”.

В правилах со знаком «–» в голове осуществляется удаление объектов из коллекции.

В языке Cypher такой запрос представляется запросом с секцией DELETE:

```
START m = node:node_auto_index(title = 'Desperado')
DELETE m
```

Пример 3 (Обновление значения атрибута). Рассмотрим запрос, устанавливающий год создания фильма «Васаби»:

```
movie(m/[year]) :-
movie(m/[title, year1/year]), m.title = “Vasabi”,
year = 2001.
```

В языке Cypher такой запрос такой запрос представляется запросом с секцией SET:

```
START m = node:node_auto_index(title = 'Vasabi')
SET m.year = 2001
RETURN year
```

4 Сохранение информации и семантики операций ЯМД при отображении

В данном разделе рассматриваются вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN [1, 4]. Применяется метод, предложенный и опробованный при унификации модели, основанной на многомерных массивах, в работе [22].

Язык AMN основан на теории множеств и типизированном языке первого порядка. Спецификации AMN называются *абстрактными машинами* и сочетают в себе пространства состояний и поведения машины, определенного операциями на состояниях. В языке AMN формализуется специальное отношение между спецификациями – *уточнение*.

Идея метода заключается в следующем. Рассмотрим исходную модель *S* и целевую модель *T*. Построим отображение θ модели *S* в модель *T*. Выразим семантику моделей в виде абстрактных машин AMN, построив при этом машины M_S и M_T соответственно. При этом структуры данных моделей представляются переменными машин, свойства структур данных представляются инвариантами машин, характерные операции моделей данных представляются операциями машин. *Операциями* в данном случае называются характерные родовые запросы в языках СИНТЕЗ и Cypher соответственно.

Рассматриваемые операции исходной и целевой модели должны быть связаны отображением ЯМД. Отображение ЯОД представляется в виде специального *склеивающего инварианта* – замкнутой формулы, связывающей состояния машин M_S и M_T.

Отображение θ считается *сохраняющим информацию и семантику операций*, если машина M_S, соответствующая исходной модели, уточняет машину M_T, соответствующую целевой модели [22]. Уточнение доказывается интерактивно при помощи специальных программных средств [4].

В качестве иллюстрации основных принципов выражения семантики синтетической графовой модели и языка СИНТЕЗ в AMN рассмотрим частичные (в связи с ограниченным объемом статьи) AMN-спецификации, соответствующие данным моделям. Нижеследующий текст организован следующим образом: приводятся последовательные части спецификации на языке AMN и сопровождаются комментариями.

Основные идеи представления семантики объектной модели языка СИНТЕЗ в языке AMN изложены в работе [22]. В настоящей статье рассматривается семантика специфических конструкций, необходимых для унификации графовых моделей.

Итак, *спецификация*, выражающая семантику объектной модели языка СИНТЕЗ, представляется в языке AMN конструкцией REFINEMENT:

```
REFINEMENT ObjectDM
```

Константы, необходимые для унификации графовой модели, объявлены в разделе CONSTANTS машины *ObjectDM* и типизируются в разделе PROPERTIES:

```
CONSTANTS
    c_edges, c_vertices,
    a_startVertex, a_endVertex,
    c_edges_instance_type
PROPERTIES …
```

Раздел PROPERTIES содержит формулу, которая состоит из предикатов, типизирующих константы Предикаты соединяются операцией конъюнкции. Так, имена классов ребер и вершин представлены константами *c_edges* и *c_vertices*, тип которых – подмножество множества строк (*STRING_Type*):

```
    c_edges: STRING_Type &
    c_vertices: STRING_Type
```

Знак типизации «:» формально означает принадлежность элемента множеству.

Имя типа экземпляров класса ребер представлено константой *c_edges_instance_type*:

```
    c_edges_instance_type: STRING_Type
```

Идентификаторы атрибутов этого типа, соответствующих исходящей и входящей вершинам ребра, представляются константами *a_startVertex, a_endVertex*, тип которых – натуральное число (*NAT*):

```
    a_startVertex: NAT &
    a_endVertex: NAT
```

Переменные, составляющие пространство состояний объектной модели, объявлены в разделе ABSTRACT_VARIABLES машины *ObjectDM* и типизируются в разделе INVARIANT:

```
ABSTRACT_VARIABLES
    m_directed, m_restricted,
    m_startVertexType,  m_endVertexType,
    isValidEdge
INVARIANT …
```

Раздел INVARIANT содержит формулу, которая состоит из предикатов, типизирующих переменные состояния, и налагающих различные совместные ограничения на переменные и константы. Предикаты соединяются операцией конъюнкции.

Так, декларируется, что *c_edges* и *c_vertices* действительно являются именами классов:

```
c_edges: classNames &
c_vertices: classNames
```

Здесь *classNames* – множество, содержащее имена всех классов базы данных [22].

Метаинформация, связанная с типом экземпляров класса ребер, представлена переменными *m_directed* (направленность ребра), *m_restricted* (определенность типов вершин ребра), *m_startVertexType* (тип исходящей вершины), *m_endVertexType* (тип входящей вершины):

```
m_directed: subclasses(c_edges) --> BOOL &
m_restricted: subclasses(c_edges) --> BOOL &
m_startVertexType:
    subclasses(c_edges) --> subclasses(c_vertices) &
m_endVertexType:
    subclasses(c_edges) --> subclasses(c_vertices)
```

Переменные типизированы полными функциями (знак -->), определенными на множестве всех классов ребер (которые являются подклассами класса всех вершин *c_edges*). Функция *subclasses* ставит в соответствие классу множество его подклассов.

Декларируется, что *c_edges_instance_type* действительно является именем типа, а атрибуты *a_startVertex* и *a_endVertex* являются атрибутами этого типа. Декларируется также, что тип значений данных атрибутов – абстрактный тип данных (*ADT*):

```
c_edges_instance_type: typeNames &
a_startVertex: typeAttributes(c_edges_instance_type) &
a_endVertex: typeAttributes(c_edges_instance_type) &
attributeType(a_startVertex) = ADT &
attributeType(a_endVertex) = ADT
```

Здесь функция *typeAttributes* возвращает множество атрибутов типа, функция *attributeType* – тип значений атрибута [22].

Предикат смежности вершин и ребер представляется функцией *isValidEdge*, сопоставляющей ребру *edg* и двум вершинам v_1, v_2 значение *истина*, если вершины v_1, v_2 соединены ребром *edg*:

```
isValidEdge: objectsOfClass(c_edges)*
    objectsOfClass(c_vertices) *
    objectsOfClass(c_vertices) --> BOOL
!(edg, v1, v2).(edg: objectsOfClass(c_edges) &
    v1: objectsOfClass(c_vertices) &
    v2: objectsOfClass(c_vertices) =>
    ((isValidEdge(edg, v1, v2) = TRUE) <=>
    (adtAttributeValue(a_startVertex)(edg) = v1 &
    adtAttributeValue(a_endVertex)(edg) = v2) ) )
```

Здесь * - знак декартова произведения множеств. Функция *adtAttributeValue(a)(o)* возвращает значение атрибута *a* объекта *o* [22].

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, ребро обязательно связывает два объекта из класса *c_vertices* (вершины):

```
!edg.(edg: objectsOfClass(c_edges) =>
    adtAttributeValue(a_startVertex)(edg):
      objectsOfClass(c_vertices) &
    adtAttributeValue(a_endVertex)(edg):
      objectsOfClass(c_vertices) )
```

Здесь «!» – знак квантора всеобщности, «=>» – логическая импликация. Функция *objectsOfClass* возвращает множество объектов – экземпляров класса [22].

Если типы вершин, соединяемых ребром, определены, то они должны принадлежать классам, задаваемым метаатрибутами *startVertexType* и *endVertexType* класса ребра:

```
!(cls, edg).(cls: subclasses(c_edges) &
    edg: objectsOfClass(cls) =>
    (m_restricted(cls) = TRUE =>
    adtAttributeValue(a_startVertex)(edg):
      objectsOfClass(m_startVertexType(cls)) &
    adtAttributeValue(a_endVertex)(edg):
      objectsOfClass(m_endVertexType(cls)) ) ) &
```

Из всего ЯМД в спецификации рассмотрена единственная операция *deleteVertex* удаления вершины:

```
OPERATIONS
deleteVertex(attr, cond) =
PRE attr : dom(attributeNames) &
  cond : INT --> BOOL &
  attributeType(attr) = Integer
THEN
  objectsOfClass(c_vertices) :=
  objectsOfClass(c_vertices) -
  { vert | vert: objectsOfClass(c_vertices) &
    vert: dom(adtAttributeValue(attr)) &
    cond(integerAttributeValue(attr)(vert)) = TRUE }
END
```

Параметрами операции являются идентификатор целочисленного атрибута *attr* и функция *cond*, отвечающая условию на значение атрибута. Операция *deleteVertex* удаляет из класса *c_vertices* все такие вершины *vert*, что на *vert* определен атрибут *attr*, и для значения этого атрибута выполнено условие *cond*. Здесь знак «:=» означает присваивание, знак «-» - разность множеств; конструкция *{v | F(v)}* – выделение множества таких значений *v*, что предикат *F(v)* обращается в истину, функция *integerAttributeValue(a)(o)* возвращает значение целочисленного атрибута *a* объекта *o*.

Спецификация, выражающая семантику синтетической графовой модели, представляется в языке AMN конструкцией

```
REFINEMENT GraphDM
```

Переменные, составляющие пространство состояний объектной модели, объявлены в разделе ABSTRACT_VARIABLES машины *GraphDM*:

```
ABSTRACT_VARIABLES
  vertexTypeIDs, edgeTypeIDs, attributeIDs,
  typeName, attributes, attributeName, attributeTyping,
  directed, restricted, headType, tailType,
  vertices, vertixType, edges, edgeType,
  headVertix, tailVertix,
  g_integerAttributeValue
```

Идентификаторы типов вершин представлены переменной *vertexTypeIDs*; идентификаторы типов ребер - переменной *edgeTypeIDs*; идентификаторы атрибутов – переменной *attributeIDs*; имена типов - переменной *typeName*; принадлежность атрибутов типам – переменной *attributes*; имена атрибутов – переменной *attributeName*; типы значений атрибутов – переменной *attributeTyping*; направленность ребер - переменной *directed*; определенность типов исходящей и входящей вершин ребра - переменными *restricted*, *headType, tailType*; вершины и ребра, составляющие базу данных - переменными *vertices, edges*; типы конкретных вершин и ребер - переменными *vertixType, edgeType;*исходящая и входящая вершины конкретных ребер - переменными *headVertix, tailVertix;* значения целочисленных атрибутов - переменной *g_integerAttributeValue*. Функции, представляющие значения атрибутов других типов (например, *BOOL* или *STRING*), определяются аналогично.

Переменные типизируются в разделе INVARIANT при помощи частичных (знак «+->») и тотальных функций аналогично переменным, использующимся для придания семантики объектной модели:

```
INVARIANT
    vertexTypeIDs: POW(NAT) &
    edgeTypeIDs: POW(NAT) &
    attributeIDs: POW(NAT) &
    typeName: vertexTypeIDs \/ edgeTypeIDs -->
       STRING_Type &
    attributes: vertexTypeIDs \/ edgeTypeIDs -->
       POW(attributeIDs) &
    directed: edgeTypeIDs --> BOOL &
    restricted: edgeTypeIDs --> BOOL &
    headType: edgeTypeIDs +-> vertexTypeIDs &
    tailType: edgeTypeIDs +-> vertexTypeIDs &
    attributeName: attributeIDs --> STRING_Type &
    attributeTyping: attributeIDs --> BuiltInTypes &
    vertices: POW(NAT) &
    vertixType: vertices --> vertexTypeIDs &
    edges: POW(NAT) &
    edgeType: edges --> edgeTypeIDs &
    headVertix: edges --> vertices &
    tailVertix: edges --> vertices &
    g_integerAttributeValue:
       (vertices \/ edges)*attributeIDs +-> INT &
```

Здесь знак «\/» означает объединение множеств.

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, для тех типов, метаатрибут *restricted* которых принимает значение *TRUE*, заданы типы исходящей и входящей вершин:

```
!(type).(type: edgeTypeIDs =>
      (restricted(type) = TRUE =>
         type: dom(headType) & type: dom(tailType)) &
      (restricted(type) = FALSE =>
         type /: dom(headType) & type /: dom(tailType)) )
```

Здесь функция *dom* возвращает область определения функции, знак «/:» означает непринадлежность элемента множеству.

Функция *g_integerAttributeValue* определена только для целочисленных атрибутов. Атрибут, для которого определена эта функция, принадлежит типу соответствующей вершины или ребра:

```
!(vert, attr).(vert: vertices & attr: attributeIDs =>
      ((vert |-> attr) : dom(g_integerAttributeValue) =>
         attributeTyping(attr) = Integer) &
         attr: attributes(vertixType(vert))  ) &
!(edg, attr).(edg: edges & attr: attributeIDs =>
      ((edg |-> attr) : dom(g_integerAttributeValue) =>
        attributeTyping(attr) = Integer) &
        attr: attributes(edgeType(edg))  )
```

Если типы вершин, соединяемых ребром, определены (значение метаатрибута *restricted* типа этого ребра принимает значение *TRUE*), то они должны принадлежать типам, задаваемым метаатрибутами *headType* и *tailType* типа ребра:

```
!edg.(edg: edges =>
      (restricted(edg) = TRUE =>
         vertixType(headVertix(edg)) =
            headType(edgeType(edg)) &
         vertixType(tailVertix(edg)) =
            tailType(edgeType(edg))  )  )
```

Аналогично объектной модели рассмотрена единственная операция ЯМД – операция удаления вершины *deleteVertex*:

```
OPERATIONS
deleteVertex(attr, cond) =
PRE attr: attributeIDs & cond: INT --> BOOL &
   attributeTyping(attr) = Integer
THEN
    vertices := vertices -
    {vert | vert: vertices &
       attr: attributes(vertixType(vert)) &
       cond(g_integerAttributeValue(vert, attr)) = TRUE }
END
```

Сигнатура операции совпадает с сигнатурой операции объектной модели. Семантика операции также аналогична: вершина *vert* удаляется из базы данных (множества *vertices*), если на *vert* определен атрибут *attr*, и для значения этого атрибута выполнено условие *cond.*

Для формального доказательства того, что машина *GraphDM* уточняет машину *ObjectDM* необходимо построить *инвариант уточнения*, связывающий переменные машин и добавить его к инварианту уточняющей машины.

Инвариант формализует принципы отображения ЯОД, изложенные в разделе 3.1 и объединяет их в одну конъюнкцию.

Множество имен типов графовой модели совпадает с множеством имен классов объектной модели (за исключением предопределенных классов *c_edges, c_vertices*):

```
ran(typeName) = classNames - {c_edges, c_vertices}
```

Множество атрибутов типов графовой модели соответствует множеству атрибутов объектной модели (за исключением предопределенных атрибутов *a_startVertex, a_endVertex*):

```
attributeIDs = dom(attributeNames) –
      {a_startVertex, a_endVertex}
```

Имена и типы атрибутов графовой и объектной модели совпадают:

```
!attr.(attr: attributeIDs =>
      attributeName(attr) = attributeNames(attr) &
      attributeTyping(attr) = attributeType(attr)  )
```

Вершины и ребра графовой базы данных соответствуют объектам классов *c_vertices* и c*_edges*:

```
vertices = objectsOfClass(c_vertices) &
edges = objectsOfClass(c_edges)
!vert.(vert: vertices =>
 ((vert: objectsOfClass(typeName(vertixType(vert)))) <=>
 (vert: vertices))  ) &
!edg.(edg: edges =>
  ((edg: objectsOfClass(typeName(edgeType(edg)))) <=>
  (edg: edges))  )
```

Значения атрибутов вершин и ребер графовой модели совпадают со значениями соответствующих атрибутов соответствующих объектов:

```
!(vert, attr).(vert: vertices & attr: attributeIDs =>
      ((vert |-> attr) : dom(g_integerAttributeValue) =>
         g_integerAttributeValue(vert, attr) =
            integerAttributeValue(attr)(vert))   )
!(edg, attr).(edg: edges & attr: attributeIDs =>
      ((edg |-> attr) : dom(g_integerAttributeValue) =>
         g_integerAttributeValue(edg, attr) =
            integerAttributeValue(attr)(edg))   )
```

Для указания того, что машина *GraphDM* уточняет машину *ObjectDM*, в машину *GraphDM* была добавлена директива

```
REFINES ObjectDM
```

Спецификации *ObjectDM* и *GraphDM* вместе с инвариантом уточнения были загружены в инструментальное средство Atelier B [4]. Автоматически были сгенерированы теоремы, выражающие уточнение спецификаций. В частности, для операции *deleteVertex* были сгенерированы 15 теорем, все они были доказаны также автоматически.

5 Родственные исследования и направления дальнейшей работы

Известно сравнительно небольшое количество работ, в которых исследуются вопросы интеграции или отображения графовых моделей данных. Например, в работе [18] язык запросов над гиперграфами используется для описания взглядов при интеграции графовых баз данных в посредниках. В работе [13] гиперграфовая модель также используется для интеграции графовых баз данных. Предлагается набор операций в рамках гиперграфовой модели для преобразования схемы ресурса в федеративную схему. В данных работах вопрос модельной неоднородности не встает, так как и в качестве канонической модели, и в качестве модели ресурсов выступает гиперграфовая модель. В работе [14] рассматривается отображение реляционной модели в гиперграфовую и императивная реализация операций реляционной алгебры в гиперграфовой модели. Таким образом, в качестве канонической модели также выступает гиперграфовая модель, а в качестве модели ресурса – реляционная.

В области интеграции графовых баз данных существует еще одна группа работ, в которых рассматриваются вопросы поглощения запросов, ответа на запросы и переписывания запросов с использованием взглядов (представлений). Текущие результаты в данной области изложены в работе [5]. Получены верхние границы сложности ответа на запросы, переписывания запросов с использованием GLAV-взглядов (Global and Local As View) в графовых моделях; доказана разрешимость поглощения запросов.

Основные особенности настоящей работы состоят в следующем. Целью работы является устранение модельной неоднородности современных графовых СУБД для дальнейшей их виртуальной или материализованной интеграции. В качестве исходной модели при отображении используется синтетическая модель, структуры данных которой покрывают возможности современных СУБД, основанных на простых и атрибутированных графах. В качестве целевой модели используется каноническая объектно-фреймовая модель - язык СИНТЕЗ. Для отображения обеспечивается формальное доказательство сохранения информации и семантики операций ЯМД.

Дальнейшая работа включает следующие этапы:

- выбор конкретных графовых моделей, основанных на простых и атрибутированных графах и построение трансформаций, реализующих изложенное отображение;
- расширение инструментальных средств поддержки предметных посредников для виртуальной интеграции графовых баз данных;
- применение технологии предметных посредников для решения научных задач в некоторой предметной области над множеством неоднородных ресурсов, включающим графовые базы данных.

Литература

[1] Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.

[2] R. Angles, C. Gutierrez. Survey of Graph Database Models. ACM Computing Surveys, Vol. 40, No. 1. Article No. 1, 2008.

[3] R. Angles. A Comparison of Current Graph Database Models. Proc. IEEE 28th International Conference on Data Engineering Workshops (ICDEW), 2012. – P. 171-177.

[4] Atelier B, the industrial tool to efficiently deploy the B Method. http://www.atelierb.eu/index-en.php

[5] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Vardi. Query Processing under GLAV Mappings for Relational and Graph Databases. VLDB 2013: 61-72 (2013)

[6] Dex User Manual. 2013. http://www.sparsity-technologies.com/downloads/UserManual.pdf

[7] B. Iordanov. Hypergraphdb: a generalized graph database. Proc. 2010 International Conference on Web-age information management (WAIM). Springer-Verlag, 2010, pp. 25–36.

[8] Kalinichenko L.A., Briukhov D.O., Martynov D.O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures. Proc. of the 9th International Conference on Enterprise Information Systems ICEIS 2007. - Funchal, 2007. - Volume Databases and Information Systems Integration. - P. 246-251.

[9] Kalinichenko L.A., Stupnikov S.A. Heterogeneous information model unification as a pre-requisite to resource schema mapping // A. D'Atri and D. Saccà (eds.), Information Systems: People, Organizations, Institutions, and Technologies (Proc. of the V Conference of the Italian Chapter of Association for Information Systems itAIS). – Berlin-Heidelberg: Springer Physica Verlag, 2010. – P. 373-380.

[10] Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. Moscow: IPI RAN, 2007. - 171 p.

[11] Neo4j Graph Database. - http://www.neo4j.org/

[12] RDF Primer. W3C Recommendation 10 February 2004. Eds. F. Manola, E. Miller. 2004. - http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[13] Srikrishnan Sundaresan, Gongzhu Hu: Schema integration of distributed databases using hyper-graph data model. IRI 2005:548-553

[14] Amani Tahat, Maurice H. T. Ling: Mapping Relational Operations onto Hypergraph Model CoRR abs/1105.6118 (2011)

[15] The Dex Graph Database Management System. http://www.sparsity-technologies.com/dex.php

[16] The Fourth Paradigm: Data-Intensive Scientific Discovery. Eds. Tony Hey, Stewart Tansley, and Kristin Tolle. Redmond: Microsoft Research, 2009.

[17] The Neo4j Manual. 2013. - http://docs.neo4j.org/

[18] Dimitri Theodoratos: Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model. BNCOD 2002:166-182

[19] P. T. Wood. Query languages for graph databases. ACM SIGMOD Record. 2012. V. 41, I. 1. P. 50-60.

[20] Захаров В. Н., Калиниченко Л. А., Соколов И. А., Ступников С. А. Конструирование канонических информационных моделей для интегрированных информационных систем // Информатика и ее применения. – М., 2007. – Т. 1, Вып. 2. – С. 15-38.

[21] Скворцов Н. А. Отображение моделей данных NoSQL в объектные спецификации. Труды RCDL'2012. – Переславль-Залесский: Университет города Переславля, 2012. С. 78-87.

[22] Ступников С. А. Унификация модели данных, основанной на многомерных массивах, при интеграции неоднородных информационных ресурсов. Труды RCDL'2012. – Переславль-Залесский: Университет города Переславля, 2012. С. 67-77.

Mapping of a Graph Data Model into a Canonical Information Model for the Heterogeneous Information Resource Integration

Sergey Stupnikov

In the paper a mapping of an attributed graph data model into an object-frame canonical information model used for virtual or materialized database integration is presented.

An aim of the work is developing of a sound theoretical basis for the integration of graph-based resources. A verification of the mapping using a formal specification language and a specific theorem prover is provided.